Introduction to:
Computers & Programming:
Object Oriented Programming (OOP)

Adam Meyers
New York University
Summary

• What is OOP?
• OOP we covered this semester?
• Modeling a Problem in OOP
• What are some advantages?
• Sample Programs
Background

• History of Object-Oriented Programming (OOP)
  – Existed in some form since 1950s
  – Popularized by C++ and Java in 1980s and 90s
• Advantages
  – Organizes code
  – Easier for others to understand complex code
  – Easier for programmers to maintain, extend, reuse
  – Easier for programmers to collaborate
• Main Disadvantage
  – May require a big program for a little task
OOP Terminology with turtle examples

- **turtle.py** (defines turtle module)
  - [https://svn.python.org/projects/python/branches/pep-0384/Lib/turtle.py](https://svn.python.org/projects/python/branches/pep-0384/Lib/turtle.py)
- **Object** = class of data structure with specified attributes and methods
- **Instance** of an object – item belonging to that class
  - `my_turtle` is an instance of `Turtle`
- **Attribute** of an object: variable associated with object
  - `my_turtle.screen` → the screen object for `my_turtle`
  - `my_turtle._position` → the x and y position of `my_turtle`, e.g., (0,0)
- **Method**: function associated with object
  - takes object as one of its arguments
  - different method with same name for different objects (have similar functions)
  - `my_turtle.fd`, `my_turtle.left`, `my_turtle.setpos`, …
  - `wiggle_turtle` (defined in `wiggle.py`) is a subclass of `turtle` with different `fd` method
    - In theory, could rewrite other methods as well
- **Classes and subclasses**
  - `Disc` and `Tower` are subclasses of `turtle` defined in `minimal_hanoi.py`
  - `wiggle_turtle` is a subclass of `turtle` defined in `wiggle.py`
  - `Turtle` is a subclass of `RawTurtle`
OOP Framework

• Objects organized in an IS-A hierarchy
  – An armchair IS-A chair; A chair IS-A piece of furniture; A piece of furniture IS-A physical object; … IS-A object (e.g., hypernyms in Princeton's WordNet)

• Similar Methods/Functions with same name
  – Analogous operations for different objects
  – The print function
    • prints the object defined by: OBJECT._str_()
      – (period, 2 underscores, str, 2 underscores)

• A layer of organization to code
  – Provides a way of setting standards
  – Easier to share code and collaborate
  – Easier to plan several similar projects
OOP Code for Dice Games

- **sample_dice_game.py** –
  - plays 3 dice games (importing other files)
- **dice_games.py** – defines dice games
  - Imports dice.py and game_utilities.py
- **dice.py** – defines dice for use in dice_games
- **game_utilities.py** – defines useful methods not specific to dice games
Dice in dice.py

- **cheat_die1** is a subclass of **die**
  - `issubclass(cheat_die1,die) -> True`
- init function of **die**, takes one parameter: **number_of_sides** (typically 6)
- init function of **cheat_die1** also takes an additional parameter: **extra_sides**
- Compare these two types of dice
  - `dice1 = die()`
  - `dice2 = cheat_die1()`
  - compare results for `dice2.roll()` with `dice1.roll()`
  - compare attributes and methods
- How do dice display when evaluated and printed?
- 3 Methods with double underscores – part of all objects in Python (defaults if not defined)
Inherited Methods in dice.py

• most methods in die are inherited by subclass cheat_die1

• cheat_die1 contains more specific definitions
  – __init__
    • includes attributes: hidden_face and extra_sides
  – roll
    • counts extra_sides as extra chances to get highest roll

• more specific definitions override general ones
  – only undefined methods are inherited
Defining a Class

• Minimum class definition assumes defaults
  
  ```python
  class game:
      None
  
  • Initialization function based on name of class
  
  • test_game = dice_game_empty()
  
  • What does object 'look' like and how does it print?
  
  • Class definition with some defaults overridden
  
  • See definition for dice_game in OOP_examples.py
  
  • redefining __init__, __repr__, __str__
  
  • __init__ adds attributes score, game_name
  
  • __repr__ and __str__ determine how this object is represented (and how it prints)
  
  • Other methods can be defined and refined for the dice_game class
  
  • See next slide for taxonomy of games
Expanding on Taxonomy in dice_games.py
Methods for Subclasses of Dice Game

- **dice_game**
  - print_score
  - one_turn – one turn of the game
  - game_over_whoWins – ends the game, identifies winner
  - play_game – repeats one_turn until game ends

- **war_dice_games**
  - separate dice and scores for user and computer
  - player who gets a minimum score wins
  - player can cheat by using a weighted die

- **single_dice_war_simple** (subclass of war_dice_games)
  - one_turn: player scores increase by roll one die each

- **single_dice_war_with_pot** (subclass of dice_game)
  - high scoring player increases score by difference between rolls
  - winning player wins pot (sum of previous tie rolls)
Implementing Expanded Ontology

• Strategy games and Card games
  – similar method definitions to dice_games
    • __repr__, __str__ and play_game
      – same methods with different definitions
        • one_turn and game_over_who_wins

• Implement other playing pieces
  – deck_of_cards, pawns, etc.
Summary

- OOP is commonly used to organize code into sets of methods and attributes
- Objects are defined in a class hierarchy
- Real differences between classes (and subclasses)
  - Different definitions of attributes and methods
  - If definition in class is different from subclass
    - Definition in subclass is used
- Not a required topic for the final exam
  - May be problems with OOP options
Introduction to Computer Science
CSCI-UA.0101

- This course (or a placement test) is a pre-requisite
  - Next course sequentially
- Teaches the Java Programming language
  - Pure OOP language (all programs must be OOP)
    - Advantage: good for large programs and collaboration
    - Disadvantages:
      - overkill for some programs (Python may be more appropriate for little programs)
      - Not as efficient as C or C++ (especially for computationally intensive programs)
  - Popular language
- Handles wider range of data structures and algorithms
- Goes deeper into OOP techniques